

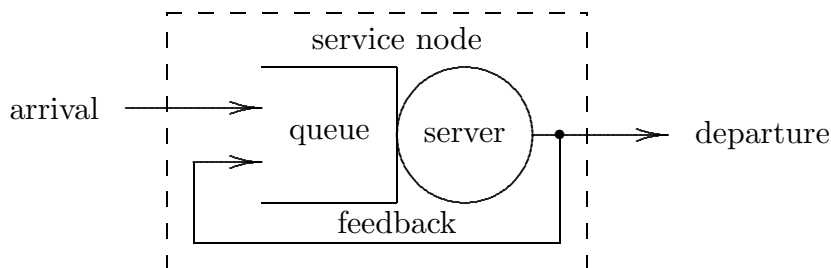
In this section we will consider three discrete-event system models, each of which is an extension of a model considered previously. The three models are (i) a single-server service node *with immediate feedback*, (ii) a simple inventory system *with delivery lag*, and (iii) a single-server machine shop.

3.3.1 SINGLE-SERVER SERVICE NODE WITH IMMEDIATE FEEDBACK

We begin by considering an important extension of the single-server service node model first introduced in Section 1.2. Consistent with the following definition (based on Definition 1.2.1) the extension is *immediate feedback* — the possibility that the service a job just received was incomplete or otherwise unsatisfactory and, if so, the job feeds back to once again request service.

Definition 3.3.1 A single-server service node *with immediate feedback* consists of a server plus its queue with a feedback mechanism, as illustrated in Figure 3.3.1.

Figure 3.3.1.
Single-server
service node
with immediate
feedback.



Jobs arrive at the service node, generally at random, seeking service. When service is provided, the time involved is generally also random. At the completion of service, jobs either depart the service node (forever) or immediately feed back and once again seek service. The service node operates as follows: as each job arrives, if the server is busy then the job enters the queue, else the job immediately enters service; as each job completes service, either a departure or a feedback occurs, generally at random. When a *departure* occurs, if the queue is empty then the server becomes idle, else another job is selected from the queue to immediately enter service. When a *feedback* occurs, if the queue is empty then the job immediately re-enters service, else the job enters the queue, after which one of the jobs in the queue is selected to immediately enter service. At any instant in time, the state of the server will either be busy or idle and the state of the queue will be either empty or not empty. If the server is idle then the queue must be empty; if the queue is not empty then the server must be busy.

Note the distinction between the two events “completion of service” and “departure”. If there is no feedback these two events are equivalent; if feedback is possible then it is important to make a distinction. When the distinction is important, the completion-of-service event is more fundamental because at the completion of service, either a departure event or a feedback event then occurs. This kind of “which event comes first” causal reasoning is important at the conceptual model-building level.

Model Considerations

When feedback occurs we assume that the job joins the queue (if any) consistent with the queue discipline. For example, if the queue discipline is FIFO then a fed-back job would receive no priority; it would join the queue at the end, in effect becoming indistinguishable from an arriving job. Of course, other feedback queue disciplines are possible, the most common of which involves assigning a priority to jobs that are fed back. If feedback is possible the default assumption in this book is that a fed-back job will join the queue consistent with the queue discipline and a new service time will be required, independent of any prior service provided. Similarly, the default assumption is that the decision to depart or feed back is random with *feedback probability* β , as illustrated in Figure 3.3.2.

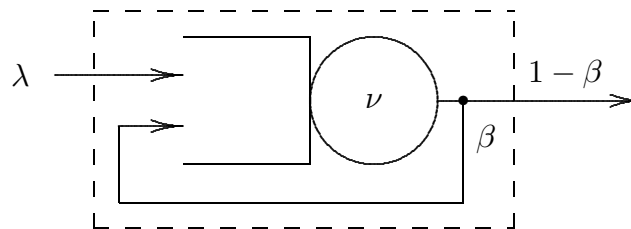


Figure 3.3.2.
Single-server
service node
with feedback
probability β .

In addition to β , the other two parameters that characterize the stochastic behavior of a single-server service node with immediate feedback are the *arrival rate* λ and the *service rate* ν . Consistent with Definition 1.2.5, $1/\lambda$ is the average interarrival time and $1/\nu$ is the average service time.*

As each job completes service, it departs the service node with probability $1 - \beta$ or feeds back with probability β . Consistent with this model, feedback is independent of past history and so a job may feed back more than once. Indeed, in theory, a job may feed back arbitrarily many times — see Exercise 3.3.1. Typically β is close to 0.0 indicating that feedback is a rare event. This is not a universal assumption, however, and so a well written discrete-event simulation program should accommodate any probability of feedback in the range $0.0 \leq \beta < 1.0$. At the computational model-building level, feedback can be modeled with a boolean-valued function as illustrated.

```
int GetFeedback(double beta)           /* use 0.0 <= beta < 1.0 */
{
    SelectStream(2);                    /* use rngs stream 2 for feedback */
    if (Random() < beta)
        return (1);                    /* feedback occurs */
    else
        return (0);                    /* no feedback */
}
```

* The use of the symbol ν to denote the service rate is non-standard. Instead, the usual convention is to use the symbol μ . See Section 8.5 for more discussion of arrival rates, service rates, and our justification for the use of ν in place of μ .

Statistical Considerations

If properly interpreted, the mathematical variables and associated definitions in Section 1.2 remain valid if immediate feedback is possible. The interpretation required is that the index $i = 1, 2, 3, \dots$ counts jobs that enter the service node; once indexed in this way, a fed-back job is not counted again. Because of this indexing all the job-averaged statistics defined in Section 1.2 remain valid *provided* delay times, wait times, and service times are incremented each time a job is fed back. For example, the average wait is the sum of the waits experienced by all the jobs that enter the service node divided by the number of such jobs; each time a job is fed back it contributes an additional wait to the sum of waits, but it does *not* cause the number of jobs to be increased. Similarly, the time-averaged statistics defined in Section 1.2 also remain valid if feedback is possible.

The key feature of immediate feedback is that jobs from outside the system are merged with jobs from the feedback process. In this way, the (steady-state) request-for-service rate is larger than λ by the positive additive factor $\beta\bar{x}\nu$. As illustrated later in Example 3.3.2, if there is no corresponding increase in service rate this increase in the request-for-service rate will cause job-averaged and time-averaged statistics to increase from their non-feedback values. This increase is intuitive — if you are entering a grocery store and the check-out queues are already long, you certainly do not want to see customers re-entering these queues because they just realized they were short-changed at check-out or forgot to buy a gallon of milk.

Note that indexing by arriving jobs will cause the average service time \bar{s} to increase as the feedback probability increases. In this case do not confuse \bar{s} with the reciprocal of the service rate; $1/\nu$ is the (theoretical) average service time *per service request*, irrespective of whether that request is by an arriving job or by a fed back job.

Algorithm and Data Structure Considerations

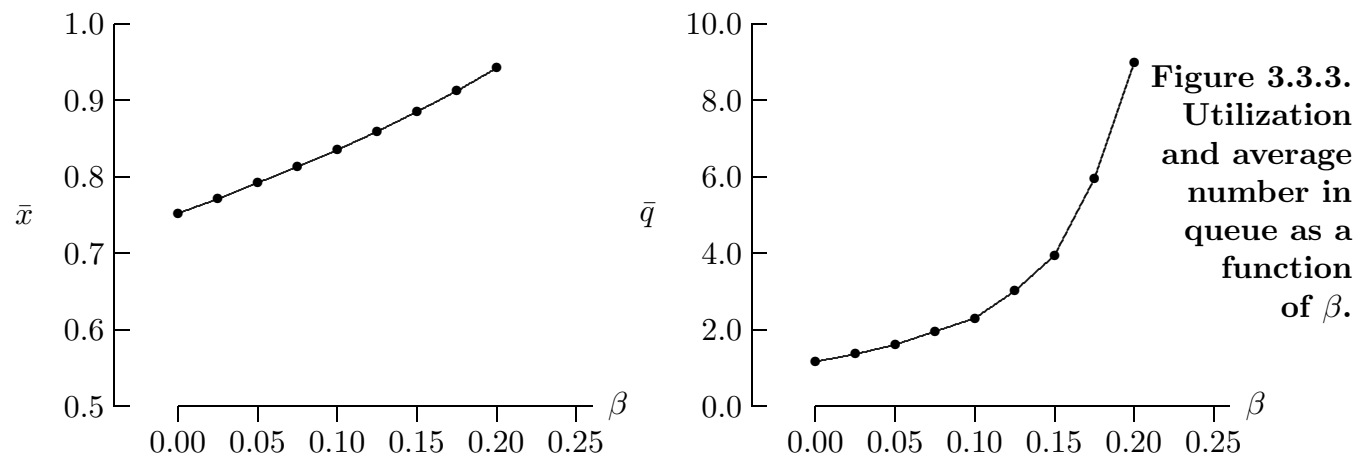
Example 3.3.1 Consider the following arrival times, service times, and completion times for the first 9 jobs entering a single-server FIFO service node with immediate feedback. (For simplicity all times are integers.)

| | | | | | | | | | | | | | | |
|------------------|---|----|-----------|-----------|----|-----------|-----------|----|-----------|----|----|-----------|-----------|-----|
| job index | : | 1 | 2 | 3 | 4 | 5 | . | 6 | . | 7 | 8 | . | 9 | ... |
| arrival/feedback | : | 1 | 3 | 4 | 7 | 10 | 13 | 14 | 15 | 19 | 24 | 26 | 30 | ... |
| service | : | 9 | 3 | 2 | 4 | 7 | 5 | 6 | 3 | 4 | 6 | 3 | 7 | ... |
| completion | : | 10 | 13 | 15 | 19 | 26 | 31 | 37 | 40 | 44 | 50 | 53 | 60 | ... |

The bold-face times correspond to jobs that were fed back. For example, the third job completed service at time 15 and immediately fed back. At the computational level, note that some algorithm and data structure is necessary to insert fed back jobs into the arrival stream. That is, an inspection of the yet-to-be-inserted feedback times (15, 26) reveals that the job fed back at time 15 must be inserted in the arrival stream after job 6 (which arrived at time 14) and before job 7 (which arrived at time 19).

The reader is encouraged to extend the specification model of a single-server service node in Section 1.2 to account for immediate feedback. Then, extend this model at the computational level by starting with program `ssq2` and using a different `rngs` stream for each stochastic process. Example 3.3.1 provides insight into an algorithmic extension and associated data structure that can be used to accomplish this.

Example 3.3.2 Program `ssq2` was modified to account for immediate feedback. Consistent with the stochastic modeling assumptions in Example 3.1.3, the arrival process has *Exponential*(2.0) random variate interarrivals corresponding to a fixed arrival rate of $\lambda = 0.50$, the service process has *Uniform*(1.0,2.0) random variate service times corresponding to a fixed service rate of $\nu \cong 0.67$ and the feedback probability is $0.0 \leq \beta \leq 1.0$. To illustrate the effect of feedback, the modified program was used to simulate the operation of a single-server service node with nine different values of levels of feedback varied from $\beta = 0.0$ (no feedback) to $\beta = 0.20$. In each case 100 000 arrivals were simulated. Utilization \bar{x} as a function of β is illustrated on the left-hand-side of Figure 3.3.3; the average number in the queue \bar{q} as a function of β is illustrated on the right-hand-side.



As the probability of feedback increases, the utilization increases from the steady-state value of $\bar{x} = 0.75$ when there is no feedback toward the maximum possible value $\bar{x} = 1.0$. If this \bar{x} versus β figure is extrapolated, it appears that saturation ($\bar{x} = 1.0$) is achieved as $\beta \rightarrow 0.25$.

Flow Balance and Saturation

The observation that saturation occurs as β approaches 0.25 is an important consistency check based on steady-state *flow balance* considerations. That is, jobs flow *into* the service node at the average rate of λ . To remain flow balanced jobs must flow *out* of the service node at the same average rate. Because the average rate at which jobs flow out of the service node is $\bar{x}(1 - \beta)\nu$, flow balance is achieved when $\lambda = \bar{x}(1 - \beta)\nu$. Saturation is achieved when $\bar{x} = 1$; this happens as $\beta \rightarrow 1 - \lambda/\nu = 0.25$. Consistent with saturation, in Example 3.3.2 we see that the average number in the queue increases dramatically as β increases, becoming effectively infinite as $\beta \rightarrow 0.25$.

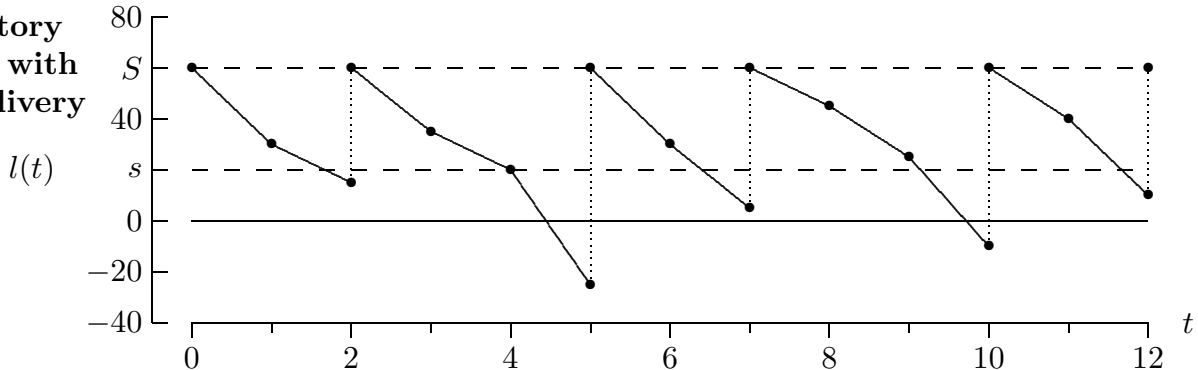
3.3.2 SIMPLE INVENTORY SYSTEM WITH DELIVERY LAG

The second discrete-event system model we will consider in this section represents an important extension of the periodic review simple inventory system model first introduced in Section 1.3. The extension is *delivery lag* (or *lead time*) — an inventory replacement order placed with the supplier will not be delivered immediately; instead, there will be a lag between the time an order is placed and the time the order is delivered. Unless stated otherwise, this lag is assumed to be random and independent of the amount ordered.

If there are no delivery lags then a typical inventory time history looks like the one in Section 1.3, reproduced in Figure 3.3.4 for convenience, with jump discontinuities possible only at the (integer-valued) times of inventory review.

Figure 3.3.4.

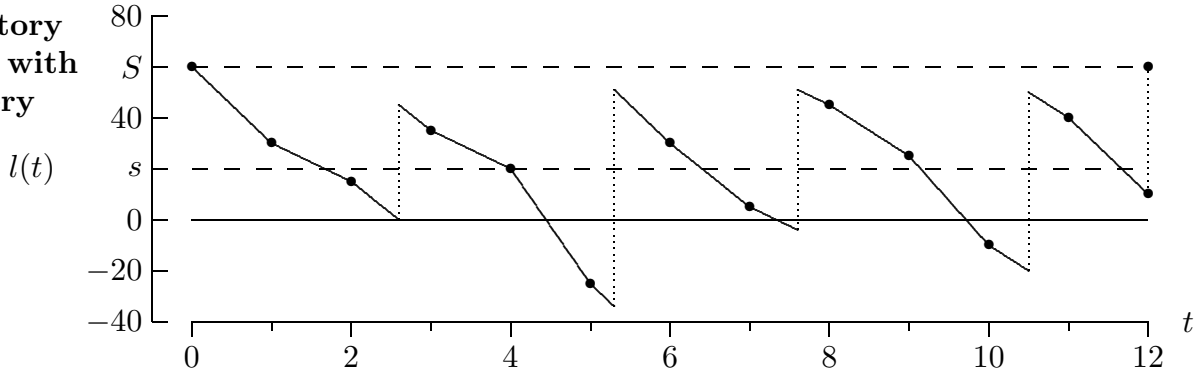
Inventory levels with no delivery lags.



If delivery lags are possible, a typical inventory time history would have jump discontinuities at arbitrary times, as illustrated in Figure 3.3.5. (The special-case order at the terminal time $t = 12$ is assumed to be delivered with zero lag.)

Figure 3.3.5.

Inventory levels with delivery lags.



Unless stated otherwise, we assume that any order placed at the beginning of a time interval (at times $t = 2, 5, 7$, and 10 in this case) will be delivered before the end of the time interval. With this assumption there is *no* change in the simple inventory system model at the specification level (see Algorithm 1.3.1). There is, however, a significant change in how the system statistics are computed. For those time intervals in which a delivery lag occurs, the time-averaged holding and shortage integrals in Section 1.3 must be modified.

Statistical Considerations

As in Section 1.3, l_{i-1} denotes the inventory level at the beginning of the i^{th} time interval (at $t = i - 1$) and d_i denotes the amount of demand during this interval. Consistent with the model in Section 1.3, the demand rate is assumed to be constant between review times. Given d_i, l_{i-1} , and this assumption, there are two cases to consider.

If $l_{i-1} \geq s$ then, because no order is placed at $t = i - 1$, the inventory decreases at a constant rate throughout the interval with no “jump” in level. The inventory level at the end of the interval is $l_{i-1} - d_i$. In this case, the equations for \bar{l}_i^+ and \bar{l}_i^- in Section 1.3 remain valid (with l_{i-1} in place of l'_{i-1} .)

If $l_{i-1} < s$ then an order is placed at $t = i - 1$ which later causes a jump in the inventory level when the order is delivered. In this case the equations for \bar{l}_i^+ and \bar{l}_i^- in Section 1.3 must be modified. That is, if $l_{i-1} < s$ then an order for $S - l_{i-1}$ items is placed at $t = i - 1$ and a *delivery lag* $0 < \delta_i < 1$ occurs during which time the inventory level drops at a constant rate to $l_{i-1} - \delta_i d_i$. When the order is delivered at $t = i - 1 + \delta_i$ the inventory level jumps to $S - \delta_i d_i$. During the remainder of the interval the inventory level drops at the same constant rate to its final level $S - d_i$ at $t = i$. All of this is summarized with the observation that, in this case, the inventory-level time history during the i^{th} time interval is defined by one vertical and two parallel lines, as illustrated in Figure 3.3.6.*

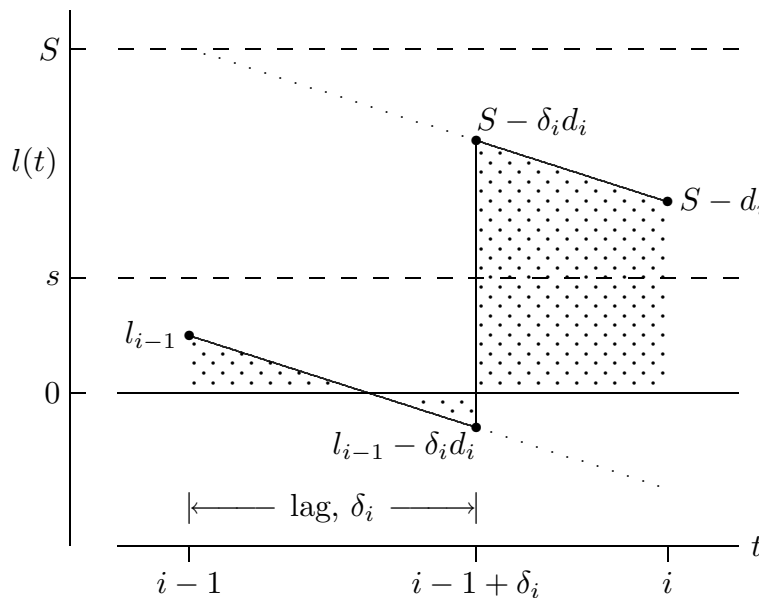


Figure 3.3.6. Inventory level during time interval i when an order is placed.

Depending on the location of the four line-segment endpoints indicated by •’s, with each location measured relative to the line $l(t) = 0$, either triangular or trapezoidal figures will be generated. To determine the time-averaged holding level \bar{l}_i^+ and time-averaged shortage level \bar{l}_i^- (see Definition 1.3.3), it is necessary to determine the area of each figure. The details are left as an exercise.

* Note that $\delta_i d_i$ must be integer-valued.

Consistency Checks

When system models are extended, it is fundamentally important to verify that the extended model is consistent with the parent model (the model before extension). This is usually accomplished by setting system parameters to special values. For example, if the feedback probability is set to zero an extended computational model that simulates a single-server service node *with* feedback reduces to a parent computational model of a single-server service node *without* feedback. At the computational level the usual way to make this kind of consistency check is to compare output system statistics and verify that, with the extension removed, the output statistics produced by the extended model agree with the output statistics produced by the parent model. Use of the library `rnsgs` facilitates this kind of comparison. In addition to these “extension removal” consistency checks, it is also good practice to check for intuitive “small-perturbation” consistency. For example, if the feedback probability is small, but non-zero, the average number in the queue should be slightly larger than its feedback-free value. The following example applies this idea to a simple inventory system model with delivery lag.

Example 3.3.3 For a simple inventory system with delivery lag we adopt the convention that δ_i is defined for all $i = 1, 2, \dots, n$ with $\delta_i = 0.0$ if and only if there is no order placed at the beginning of the i^{th} time interval (that is, if $l_{i-1} \geq s$). If an order is placed then $0.0 < \delta_i < 1.0$. With this convention the stochastic time evolution of a simple inventory system with delivery lag is driven by the two n -point stochastic sequences d_1, d_2, \dots, d_n and $\delta_1, \delta_2, \dots, \delta_n$. The simple inventory system is *lag-free* if and only if $\delta_i = 0.0$ for all $i = 1, 2, \dots, n$; if $\delta_i > 0.0$ for at least one i then the system is not lag-free. Relative to the five system statistics in Section 1.3, if the inventory parameters (S, s) are fixed then, even if the delivery lags are small, the following points are valid.

- The average order \bar{o} , average demand \bar{d} , and relative frequency of setups \bar{u} are exactly the same whether the system is lag-free or not.
- Compared to the lag-free value, if the system is not lag-free then the time-averaged holding level \bar{l}^+ will decrease.
- Compared to the lag-free value, if the system is not lag-free then the time-averaged shortage level \bar{l}^- will either remain unchanged or it will increase.

At the computational level these three points provide valuable consistency checks for a simple inventory system discrete-event simulation program.

Delivery Lag

If the statistics-gathering logic in program `sis2` is modified to be consistent with the previous discussion, then the resulting program will provide a computational model of a simple inventory system with delivery lag. To complete this modification, a stochastic model of delivery lag is needed. In the absence of information to the contrary, we assume that each delivery lag is an independent *Uniform*(0, 1) random variate.

Example 3.3.4 Program `sis2` was modified to account for $Uniform(0, 1)$ random variate delivery lags, independent of the size of the order. As an extension of the automobile dealership example (Example 3.1.7), this modified program was used to study the effect of delivery lag. That is, with $S = 80$ the average weekly cost was computed for a range of inventory threshold values s between 20 and 60. To avoid clutter only steady-state cost estimates (based on $n = 10\,000$ time intervals) are illustrated. For comparison, the corresponding lag-free cost values from Example 3.1.7 are also illustrated in Figure 3.3.7.

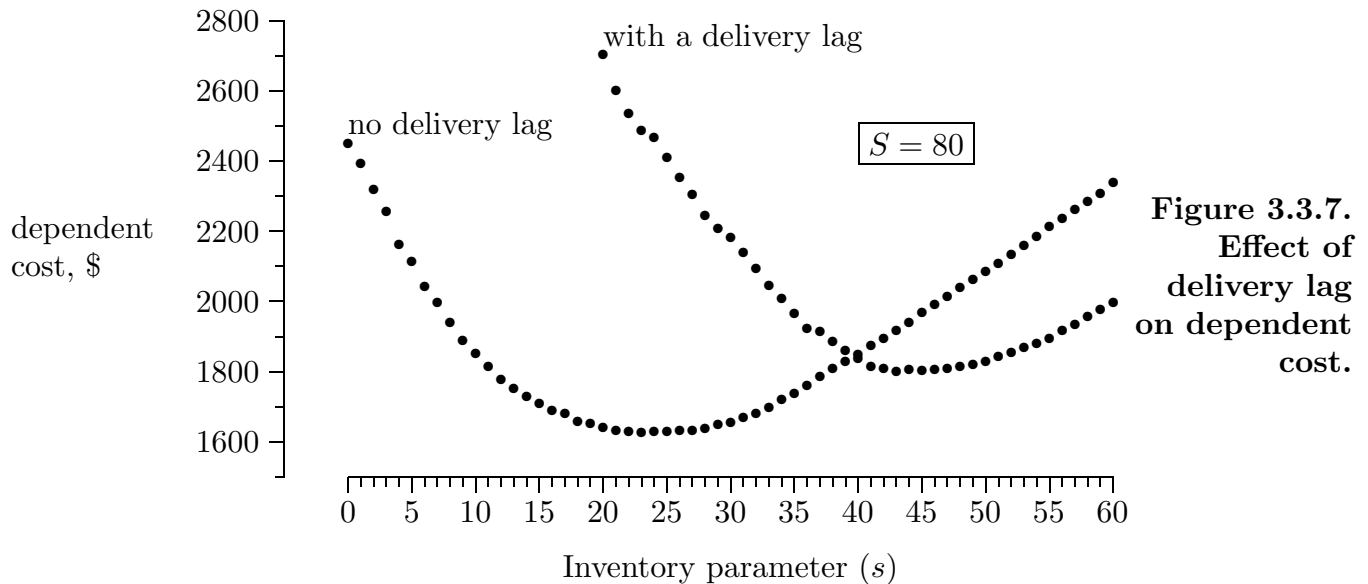


Figure 3.3.7.
Effect of
delivery lag
on dependent
cost.

Figure 3.3.7 shows that the effect of delivery lag is profound; the U-shaped cost-versus- s curve is shifted up and to the right. Because of this shift the optimum (minimum cost) value of s is increased by approximately 20 automobiles and the corresponding minimum weekly cost is increased by almost \$200.*

The shift in the U-shaped curve in Example 3.3.4 is consistent with the second and third points in Example 3.3.3. That is, delivery lags cause \bar{l}^+ to decrease and \bar{l}^- to increase (or remain the same). Because the holding cost coefficient is $C_{\text{hold}} = \$25$ and the shortage cost coefficient is $C_{\text{short}} = \$700$ (see Example 1.3.5), delivery lags will cause holding costs to decrease a little for all values of s and will cause shortage costs to increase a lot, but only for small values of s . The shift in the U-shaped curve is the result.

Examples 3.3.2 and 3.3.4 present results corresponding to significant extensions of the two canonical system models used throughout this book. The reader is strongly encouraged to work through the details of these extensions at the computational level and reproduce the results in Examples 3.3.2 and 3.3.4.

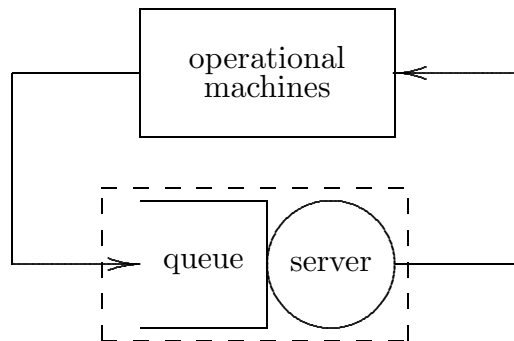
* Because of the dramatic shift in the optimum value of s from the lag-free value of $s \cong 23$ to the with-lag value of $s \cong 43$, we see that the optimal value of s is *not* robust with respect to model assumptions about the delivery lag.

3.3.3 SINGLE-SERVER MACHINE SHOP

The third discrete-event simulation model considered in this section is a single-server machine shop. This is a simplified version of the multi-server machine shop model used in Section 1.1 to illustrate model building at the conceptual, specification, and computational level (see Example 1.1.1).

A single-server machine shop model is essentially identical to the single-server service node model first introduced in Section 1.2, except for one important difference. The service node model is *open* in the sense that an effectively infinite number of jobs are available to arrive from “outside” the system and, after service is complete, return to the “outside”. In contrast, the machine shop model is *closed* because there are a finite number of machines (jobs) that are part of the system — as illustrated in Figure 3.3.8, there is no “outside”.

Figure 3.3.8.
Single-server
machine shop
system diagram.



In more detail, there is a finite population of statistically identical machines, all of which are initially in an *operational* state (so the server is initially idle and the queue is empty). Over time these machines fail, independently, at which time they enter a *broken* state and request repair service at the single-server service node.* Once repaired, a machine immediately re-enters an operational state and remains in this state until it fails again. Machines are repaired in the order in which they fail, without interruption. Correspondingly, the queue discipline is FIFO, non-preemptive and conservative. There is no feedback.

To make the single-server machine shop model specific, we assume that the service (repair) time is a *Uniform*(1.0, 2.0) random variate, that there are M machines, and that the time a machine spends in the operational state is an *Exponential*(100.0) random variate. All times are in hours. Based on 100 000 simulated machine failures, we want to estimate the steady-state time-averaged number of operational machines and the server utilization as a function of M .

* Conceptually the machines move along the network arcs indicated, from the operational pool into and out of service and then back to the operational pool. In practice, the machines are usually stationary and the server moves to the machines. The time, if any, for the server to move from one machine to another is part of the service time.

Program ssms

Program `ssms` simulates the single-server machine shop described in this section. This program is similar to program `ssq2`, but with two important differences.

- The library `rngs` is used to provide an independent source of random numbers to both the simulated machine failure process and the machine repair process.
- The failure process is defined by the array `failure` which represents the time of next failure for each of the M machines.

The time-of-next-failure list (array) is not maintained in sorted order and so it must be searched completely each time a machine failure is simulated. The efficiency of this $O(M)$ search could be a problem for large M . Exercise 3.3.7 investigates computational efficiency improvements associated with an alternative algorithm and associated data structure.

Example 3.3.5 Because the time-averaged number in the service node \bar{l} represents the time-averaged number of broken machines, $M - \bar{l}$ represents the time-averaged number of operational machines. Program `ssms` was used to estimate $M - \bar{l}$ for values of M between 20 and 100, as illustrated in Figure 3.3.9.

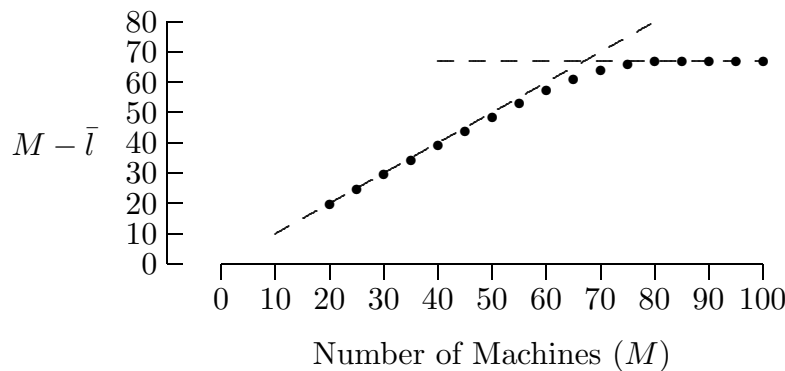


Figure 3.3.9.
Time-averaged number of operational machines as a function of the number of machines.

As expected, for small values of M the time-averaged number of operational machines is essentially M . This is consistent with low server utilization and a correspondingly small value of \bar{l} . The angled dashed line indicates the ideal situation where all machines are continuously operational (i.e., $\bar{l} = 0$). Also, as expected, for large values of M the time-averaged number of operational machines is essentially constant, independent of M . This is consistent with a saturated server (utilization of 1.0) and a correspondingly large value of \bar{l} . The horizontal dashed line indicates that this saturated-server constant value is approximately 67.

Distribution Parameters

The parameters used in the distributions in the models presented in this section, e.g., $\mu = 2.0$ for the average interarrival time and $\beta = 0.20$ for the feedback probability in the single-server service node with feedback, have been drawn from thin air. This has been done in order to focus on the simulation modeling and associated algorithms. Chapter 9 on “Input Modeling” focuses on techniques for estimating realistic parameters from data.

3.3.4 EXERCISES

Exercise 3.3.1 Let β be the probability of feedback and let the integer-valued random variable X be the number of times a job feeds back. (a) For $x = 0, 1, 2, \dots$ what is $\Pr(X = x)$? (b) How does this relate to the discussion of acceptance/rejection in Section 2.3?

Exercise 3.3.2^a (a) Relative to Example 3.3.2, based on 1 000 000 arrivals, generate a table of \bar{x} and \bar{q} values for β from 0.00 to 0.24 in steps of 0.02. (b) What data structure did you use and why? (c) Discuss how external arrivals are merged with fed back jobs.

Exercise 3.3.3 For the model of a single-server service node with feedback presented in this section, there is nothing to prevent a fed-back job from colliding with an arriving job. Is this a model deficiency that needs to be fixed and, if so, how would you do it?

Exercise 3.3.4^a Modify program `ssq2` to account for a *finite* service node capacity. (a) For capacities of 1, 2, 3, 4, 5, and 6 construct a table of the estimated steady-state probability of rejection. (b) Also, construct a similar table if the service time distribution is changed to be *Uniform*(1.0, 3.0). (c) Comment on how the probability of rejection depends on the service process. (d) How did you convince yourself these tables are correct?

Exercise 3.3.5^a Verify that the results in Example 3.3.4 are correct. Provide a table of values corresponding to the figure in this example.

Exercise 3.3.6 (a) Relative to Example 3.3.5, construct a figure or table illustrating how \bar{x} (utilization) depends on M . (b) If you extrapolate linearly from small values of M , at what value of M will saturation ($\bar{x} = 1$) occur? (c) Can you provide an empirical argument or equation to justify this value?

Exercise 3.3.7^a In program `ssms` the time-of-next-failure list (array) is not maintained in sorted order and so the list must be searched completely each time another machine failure is simulated. As an alternative, implement an algorithm and associated sorted data structure to determine if a significant improvement in computational efficiency can be obtained. (You may need to simulate a huge number of machine failures to get an accurate estimate of computational efficiency improvement.)

Exercise 3.3.8 (a) Relative to Example 3.3.2, compare a FIFO queue discipline with a priority queue discipline where fed-back jobs go the head of the queue (i.e., re-enter service immediately). (b) Is the following conjecture true or false: although statistics for the fed-back jobs change, system statistics do not change?

Exercise 3.3.9^a (a) Repeat Exercise 3.3.7 using $M = 120$ machines, with the time a machine spends in the operational state increased to an *Exponential*(200.0) random variate. (b) Use $M = 180$ machines with the time spent in the operational increased accordingly. (c) What does the $O(\cdot)$ computational complexity of your algorithm seem to be?